AD-A051 890 STANFORD UNIV CALIF DIGITAL SYSTEMS LAB F/G 9/2
THE DESIGN AND IMPLEMENTATION OF AN OPERATING SYSTEM TRACER.(U)
OCT 77 D J ROSSETTI, T H BREDT N00014-67-A-0112-0044
UNCLASSIFIED DSL-TN-97 NL

1 OF 1
AD
A051 890

END
DATE
FILMED
4-78
DDC

Center for Reliable Computing

THE DESIGN AND IMPLEMENTATION OF AN OPERATING SYSTEM TRACER

David J. Rossetti
and
Thomas H. Bredt

Technical Note No. 97

October 1977

CENTER FOR RELIABLE COMPUTING
Digital Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California 94305

This research was supported in part by the Joint Services Electronics Program under Contract N00014-67-A-0112-0044. Computing resources were provided by the SLAC Computing Services, Stanford Center for Information Processing.

The Design and Implementation of an Operating System Tracer

David J. Rossetti
and
Thomas H. Bredt

Technical Note No. 97

October 1977

Digital Systems Laboratory
Stanford University
Stanford, California 94305

ABSTRACT

STRAP, a general system for collecting a trace of machine-instruction execution on a high performance processor, has been implemented. Its generality lies in its ability to trace all instructions executed by the processor, whether they are issued by the supervisor or by a user program, and also in its ability to do this without significant impact on system performance. STRAP creates a "virtual machine" environment in which the operating system appears to be executing normally, but is actually having its instructions traced by a program lying between it and the real processor. Burst sampling is used to avoid excessive tracing overhead, and the real processor interprets the traced instructions, keeping the instantaneous burst overhead down to about 45:1.

The resulting traces, which can be useful for architectural studies and performance evaluation of existing systems, can now contain data for all parts of the operating system. We present example results showing how the data have been used to study: 1) The usage of various operating system components, 2) The difference in branching patterns between supervisor and user code, and 3) The difference in instruction frequency distribution between the two modes of operation.

INDEX TERMS: computer architecture, instruction set design, IBM 370, measurement, operating systems, performance evaluation, tracing, virtual machine

The Design and Implementation of an Operating System Tracer

David J. Rossetti

and

Thomas H. Bredt

Digital Systems Laboratory

Stanford University

INTRODUCTION

In this paper we discuss the design and implementation of a program to trace operating system and user programs. The program, called STRAP/370, runs as a job under the IBM OS/VS2 operating system and produces an instruction-by-instruction trace of code executed by the IBM 370 processor. There are two factors which make this program different from most other tracers: 1) There is no limitation on the type of program traced (e.g. supervisor mode as well as user mode instructions are traced). 2) The trace does

----

This research was supported in part by the Joint Services Electronics Program (U.S. Army, U.S. Navy, and U.S. Air Force) under Contract N00014-67-A-0112-044. Computing resources were provided by the SLAC Computing Services, Stanford Center for Information Processing.

not represent continuous instruction execution, but rather periodically samples short bursts of execution, using the intermediate periods to write trace buffers to an external medium such as tape or disk.

Significant results have been obtained in the past using program traces to evaluate computer architectures. There are many examples in the literature of trace-driven simulation projects [1, 2, 3], in which tracing has been used to provide the empirical backing for analytic methods in system evaluation. Winder [4] describes a project at RCA Labs to develop a complete library of trace tapes which were mainly applied to cache system evaluation. In the paper he describes three trace programs, one of which could trace some of the supervisor routines but was used to study only cache performance. More recently, a thesis by Lunde [5] demonstrates the evaluation of instruction sets using problem program trace data.

Previously, however, there has not been a practical way to gather complete detailed information about operating system programs. Such data could be used to measure the behavior of supervisory programs, to measure the usage of various operating system services, and to generally evaluate software system architecture. It simply has not been practical for a program to simulate or trace the entire workings of a machine such as the System/370. STRAP/370 is a program that provides the detailed trace information without resorting to wholesale interpretation.

It can trace code executing in supervisor or user (problem) state, while interrupts are enabled or disabled -- in short any program. It has been implemented on an IBM 370 Model 168 at the Stanford Linear Accelerator Center (SLAC) triplex computer system [6], running under the IBM OS/VS2 operating system. The environment and structure of STRAP/370 are discussed, then an example is given to show the applicability of this tool to operating system measurement.

THE PROBLEM

Current operating system measurement tools generally provide the ability to collect information at two levels of detail. At the job level events can be measured which could probably be best described as accounting data. As an example, the IBM System Management Facilities feature [7], an integral part of OS/VS, acts as a central agency within the system for the recording of job, and resource transactions as they occur. Gross statistics such as run time, number of tapes used, files used, etc. are available and can be used in performance measurement as well as accounting applications.

The second level of measurement detail is exemplified by the class of software monitors currently being used to tune large scale systems. The IBM Generalized Trace Facility (GTF) [8] and the Configuration Utilization Evaluator (CUE) from Boole & Babbage, Inc. [9] are examples. Monitors such as these can record system related events, such as interruptions, page faults, input/output device activity, user program activation, etc. Current operating systems and monitors provide these services either statically (as GTF does, being part of the operating system) or dynamically (as most software monitors do), through the use of traps and calls to event recording routines.

The finest level of detailed information available to a software probe relates to instruction execution. Trace programs, typically interpretive, exist for most computer systems. With slowdown factors on the order of 20:1 to 100:1 these programs simulate the hardware available to user programs. Such trace programs, however, have been able to trace only instructions executed in the problem (or user) state. All supervisor processing done on behalf of user programs is missing from the trace. In fact, since usually the tracer is itself a problem program, not even interruptions and suspensions of execution can be measured directly. Thus, classical tracing has been of value only at the local level in such areas as debugging and code optimization, and not of significant use in measuring global system behavior.

But how can more complete detailed information be obtained from a software monitor? Suppose we would like to fill the gap between current software monitors and hardware monitors by providing the ability to trace system events at the level of instruction execution. The STRAP/370 system is a system which has been developed to provide this more powerful tracing method. Any program running in the OS/VS 370 environment can be traced, including all components of the operating system itself. Thus, this new measurement facility extends the range of software tools which can be applied in the measurement of large scale systems -- from accounting data to a trace of each instruction executed.

REQUIREMENTS OF THE TRACING SYSTEM

What are the requirements of a usable tracing system? First let us consider the type of information that is required by a facility capable of monitoring supervisor and user states, possibly during interrupt handling. Events occuring at the machine language level, along with relevant state information, are listed below.

EVENT	STATE INFORMATION
Instruction execution	The instruction executed
	Operand values
	Virtual addresses, Real addresses
	Special informaion (e.g. condition code)
Interruption	Old/new processor status
	Time of occurrence
	Interruption description

Processor state change New procesor status (e.g. masks etc.)
Register contents

The first requirement is that the measurement tool be able to record the above events so that the execution of the traced segment of code can be reconstructed if desired.

Secondly, the tracing system must allow for selective recording of trace data. If, for example, a user were studying just the stream of real addresses generated by the system, then it should be possible to specify that only real addresses be recorded.

The third requirement is that the tracing facility must not noticeably degrade system performance. Clearly, there will be some overhead in the gathering of the trace data, however to measure an actual system over an extended period of time (on the order of hours), this impact must be minimal, else the use of the tool becomes impractical. It also seems clear that as degradation increases, the validity of the operating system trace data becomes more questionable. For example, external events that are asynchronous to processor execution will seem to occur at a higher rate. In an extreme example the operating system could spend most of the available time handling interrupts, due to tracing slowdown. To solve this and other problems a scheme for sampling the instruction stream in a "skip-trace" fashion is used in STRAP. The sampling method is described in the next section.

The final, most important, and most difficult requirement of a program that takes complete control of the operating system is that it be absolutely error free. Since it is running at a logical level between the operating system and the hardware, such a program must be more reliable than the operating system itself. As an example of the problems inherent in such a venture, one hardware and one operating system "bug" have been uncovered during the testing phase of the STRAP/370 system.

THE SAMPLING APPROACH

In order to prevent unacceptable system degradation, it is necessary to sample the event (instruction) stream. Each sample consists of a packet of contiguous events (mostly instructions) that are recorded until a specified amount of time, t, on the order of milliseconds, has passed. The time between samples, T, is also controlled and is on the order of seconds. Figure 1 illustrates

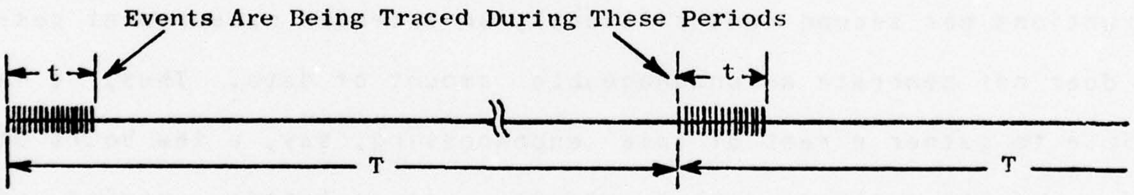

Figure 1. Periodic Sampling of the Event Stream.

this scheme. Typically, if a sample is taken every other second (T = 2 sec.) and each sample represents, say, one millisecond (t =

0.001 sec.) of execution on the IBM 370/168, less than 2 percent degradation in processor performance will result due to tracing. In addition to the benefit of control over system impact, this sampling approach affords aid in two other areas:

1) The sample can be accumulated in a memory buffer (during which time the effective execution rate is lower). At the end of the sample the operating system is allowed to run normally, thereby providing standard input/output facilities for external recording of the trace buffer. Without sampling, the system would have to be stopped at some point, since the central processor can fill buffers faster than they can be emptied; execution in this mode would surely destroy both system performance and trace data validity. An average of about 10 (8-bit) bytes of trace data is recorded for each instruction traced. The trace buffer may be as large as desired, however a reasonable size seems to be between 4,000 to 20,000 bytes.

2) Since sampling gathers on the order of thousands of instructions per second (not millions), an extended measurement session does not generate an unmanageable amount of data. Thus, it is possible to gather a reel of tape encompassing, say, a few hours of running and to analyze system activity over a longer period of time.

In spite of its advantages, the sampling method also raises a question: How is STRAP tracing to be triggered so that the samples are taken totally independent of where the system is executing? At first, one might consider using a clock, such as the hardware clock or a pseudo-clock maintained by the operating system. This use of the "timer run-out" feature is described by Cantrell and Ellison [10] in their measurement of the GE-635 system. But it is clear that an interrupt from such a source could not be allowed to occur in many critical areas of the operating system. A convincing example of such a critical routine is the timer interrupt handler; a second timer interrupt could never be tolerated there. So no interrupt controlled by the operating system can be used as a trigger if we require the trace to begin with an arbitrary instruction. Also, to get a truly random sample of instructions from the event stream, the trigger should operate asynchronously, completely isolated from events occuring within the processor. What is evidently needed is an external, independent source for trace trigger interruptions, with the requirement that such interruptions be non-maskable, i.e. always processed at the end of the current instruction.

In System/370 (S/370) there are two classes of asynchronous, non-maskable interrupt which could be used to signal the start of a trace -- Machine Check, and Restart. The Machine Check interruption occurs when a serious machine error is detected during the execution of an instruction. The processor may or may not be left in a well-defined state when the Machine Check interrupt handling routine is entered. If it were to be used for tracing purposes, a means for causing a false Machine-Check would probably be necessary. Needless to say, this approach seems a bit drastic; let us consider the other alternative.

The Restart interrupt is initiated by pressing the RESTART pushbutton on the S/370 console. It is impossible to disable under program control and its priority is such that the Restart interrupt handler is executed before any other interrupt handler, should other interrupts occur simultaneously. Thus, the Restart interrupt is an ideal candidate, since its use seems to minimize undesirable side effects, and since OS/VS makes only slight use of the restart function. *

HARDWARE ENVIRONMENT

In order to capture all system events which occur, there must be a means for gathering the machine state after each such event -- instruction, interruption, etc. One way to do this would be to construct a complete machine interpreter which would be able to simulate all user mode, supervisor mode, and external events, just as the hardware does. Another scheme would be to cause execution to be interrupted after each relevant event, in this case after each instruction is executed. Clearly, if such a "trapping" mechanism were available in the hardware, the monumental task of

----

* Restart is also used by the IBM Dynamic Support System (DSS). This subsystem is an OS/VS maintenance aid and is not used during normal operation.

emulating a full S/370 is reduced to the not so monumental task of handling these traps and collecting the state of the machine. In the S/370 such a flexible facility does exist and is called the Program Event Recording (PER) feature. In the next few paragraphs the three hardware components used by STRAP will be described. The first two, Restart and PER, are an integral part of System/370, the last, the Trace Ace, was developed specifically for this project.

Hardware interrupt processing which takes place in System/370 [11] is much like that which is done in System/360, the major difference being the manner in which fixed storage locations provide and receive certain state information. The majority of this state onformation is coalesced into a 64-bit doubleword termed the Program Status Word (PSW). It contains such program state items as: *instruction* address, storage protection key, interruption disable masks, etc. Corresponding to each interrupt there is a "new PSW" in a fixed storage location which is used to define the entry address and state of the handler for that interrupt. If, during the execution of an instruction, there is at least one interruption active for which the processor is enabled, the following sequence of events transpires:

1. Complete the processing of the current instruction.

2. Store the current Program Status Word (PSW) for the highest priority interrupt currently present in its predefined (fixed) storage location.

3. Fetch a "new PSW" corresponding to that interrupt (from its predefined storage location) and make it the current PSW, i.e. use it to define the new processor state.

4. Using the current PSW, are any enabled (unmasked) interruptions active? If yes, continue from 2; else begin execution of the interrupt handler for this interruption using the current PSW.

The net result of the above is that an interrupt handler begins execution, while all other accepted (non-masked) interruptions have been "stacked" in the chain of PSWs by the loop above. Interruptions that are not accepted (i.e. due to masking) remain active and waiting.

The Restart interrupt is used by STRAP to receive the stimuli from the external world. As alluded to earlier, its advantages are: 1) It is unmaskable, i.e. will interrupt any program. 2) Priorities are arranged such that the Restart interrupt handler will gain control first if more than one interruption is active. This is vital to STRAP's ability to trace all OS/VS interrupt handling. 3) The interrupt is easily activated by the Restart console pushbutton, or a suitable substitute.

When a Restart interrupt occurs, the PSW in storage location 0 (Restart "new PSW") is made the current PSW, while the current PSW is stored in location 8 (Restart "old PSW"). Since beforehand STRAP has judiciously tampered with the PSW in location 0, the STRAP Restart interrupt handler receives control instead of one provided by OS/VS. The Restart "new PSW" is said to be "owned" by STRAP. The STRAP Restart interrupt handler initializes the trace buffer and various flags, then prepares the S/370 Program Event Recording machinery (described next) for tracing before returning to the interrupted program. Note that this process is completely transparent to the program which was interrupted.

The PER facility uses the S/370 program interruption to sense or "trap" events associated with instruction execution. Those events which can be trapped are:

1. Successful branch - when a branch instruction actually causes transfer of control.

2. Instruction fetch - when an instruction is fetched from the virtual storage area specified by the PER bounds registers.*

3. Storage Alteration - when a storage location within the virtual storage area specified by the PER bounds

----

* The PER bounds registers are two processor registers which define, respectively, the lower and upper vitrual addresses of the storage area to be monitored. STRAP sets this range to include all addressable storage.

registers is changed.

4. General register alteration - when any of a subset of the 16 general purpose registers is modified. The subset is specified by a 16-bit mask contained in a CPU control register.

As illustrated in Figure 2 below, a PER event causes a program interruption to be taken, and in addition information describing the event is stored in fixed storage locations. In Step 1, four actions take place: 1a) The current PSW is stored in the program old PSW location as for a normal program interruption, 1b) the program interruption code is set to indicate a program event, 1c) the PER code is set to describe the nature of the event, and 1d)

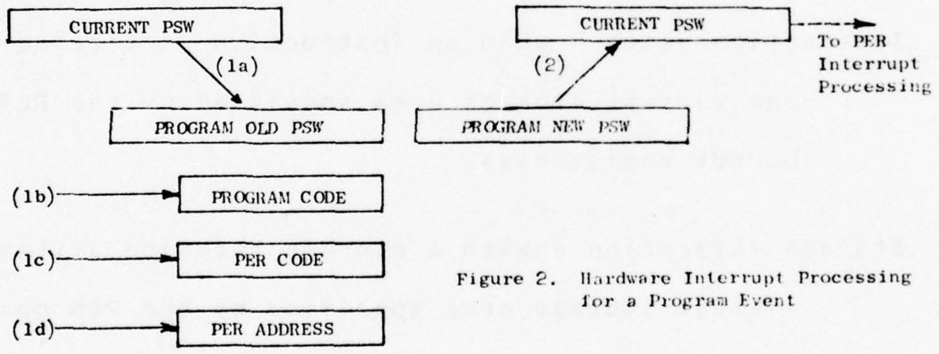

Figure 2. Hardware Interrupt Processing for a Program Event

the PER address is set to the address of the event-causing instruction. With this information, the program event handler has all it needs to analyze the situation. Step 2 involves the normal PSW loading at interrupt time. The program interrupt new PSW is made current and an interrupt handler, in our case the STRAP interrupt handler, begins execution.

The final hardware component to be discussed is the Trace Ace, the device used to generate Restart interrupts periodically, independent of the central processor. The Trace Ace is a variable-rate clock which can be set to tick at intervals from 1 to 200 seconds; each tick causes one Restart interrupt. It interfaces to the 370/168 operator control panel via a very simple connection, and as such it acts as a physical extension of the Restart pushbutton. Also, since it is completely self-contained and trivially connected, the STRAP system is highly portable, allowing it to be used in a variety of processing environments.

THE STRUCTURE AND HOW IT WORKS

This section defines the OS/VS software environment which exists prior to tracing, and shows how STRAP modifies this environment. It then describes the internal structure of STRAP and shows in general how events are processed.

In our context, the operating system acts as a buffer between the user programs and the hardware. It provides a "virtual machine" to the user which is similar to the actual hardware, but has added capability in some areas, e.g. supervisor calls, and diminished capability in others, e.g. privileged instructions. The net effect of such a transformation is an OS/VS machine which is more useful, powerful, and reliable than a "bare" system. This hierarchical nesting of real and virtual machines is illustrated in

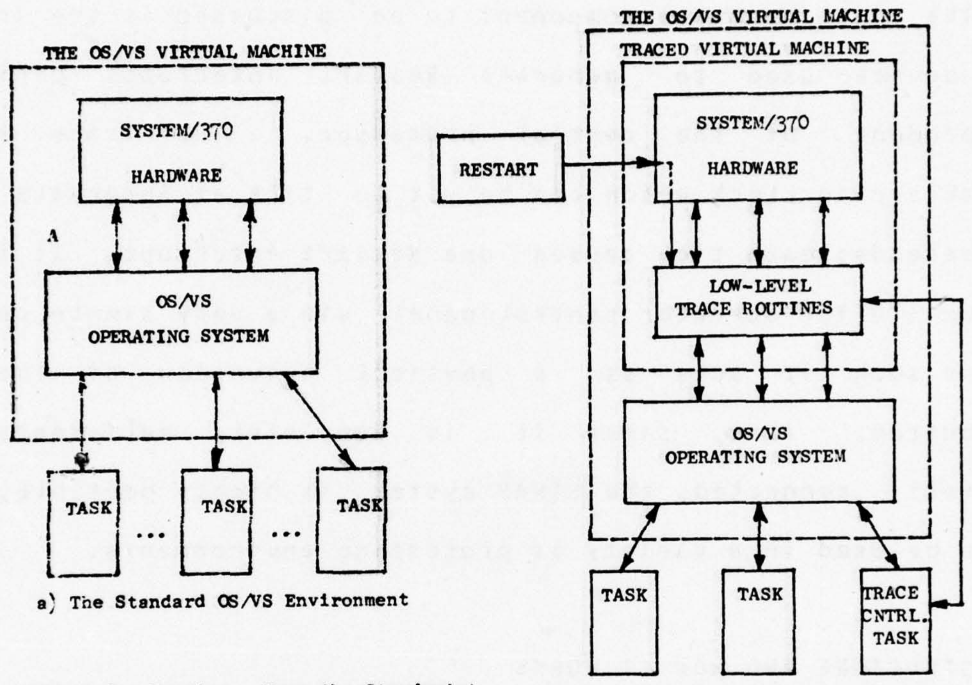

Figure 3. Tracing - From the Standpoint of Virtual Machines.

Figure 3a.

Now, if one wishes to trace all events at interface "A" in the figure above, then the operating system must be presented with a virtual machine which is identical in every way to a bare S/370. The primary goal of the STRAP system is to do just that. Using the hardware facilities described above, it has been possible to trick OS/VS into thinking it is controlling a real S/370. In actuality the real machine has been modified into one which interrupts at the end of each instruction, as shown in Figure 3b above.

The S/370 hardware has been logically replaced by another virtual machine which simulates S/370 to the operating system. What the hardware actually executes is an instruction stream composed of OS/VS instructions intermixed with the STRAP instructions needed to gather data. Again it will be stressed that the operating system cannot tell the difference, except for a momentary slowdown while the sample (on the order of a thousand instructions) is being collected. In fact, all modifications are made at run time and relate just to the fixed storage areas maintained by the hardware, allowing STRAP to be portable to any similar operating system, with no installation changes required. In summary, a clean, hierarchical structure has been established so that system modifications necessary for tracing are simple and localized to the hardware/software interface.

A more detailed representation of the internal structure of STRAP is given in Figure 4. As shown, all possible communication paths between OS/VS and the hardware have, by necessity, been "virtualized" by various components of STRAP. The zeroth-level interrupt handlers (ZLIH) provide the means for recording interruptions before they are seen by the OS/VS first-level interrupt handlers (FLIH). The ZLIHs deposit information into the trace buffer, then effectively simulate the event so that normal OS/VS interrupt processing may take place.

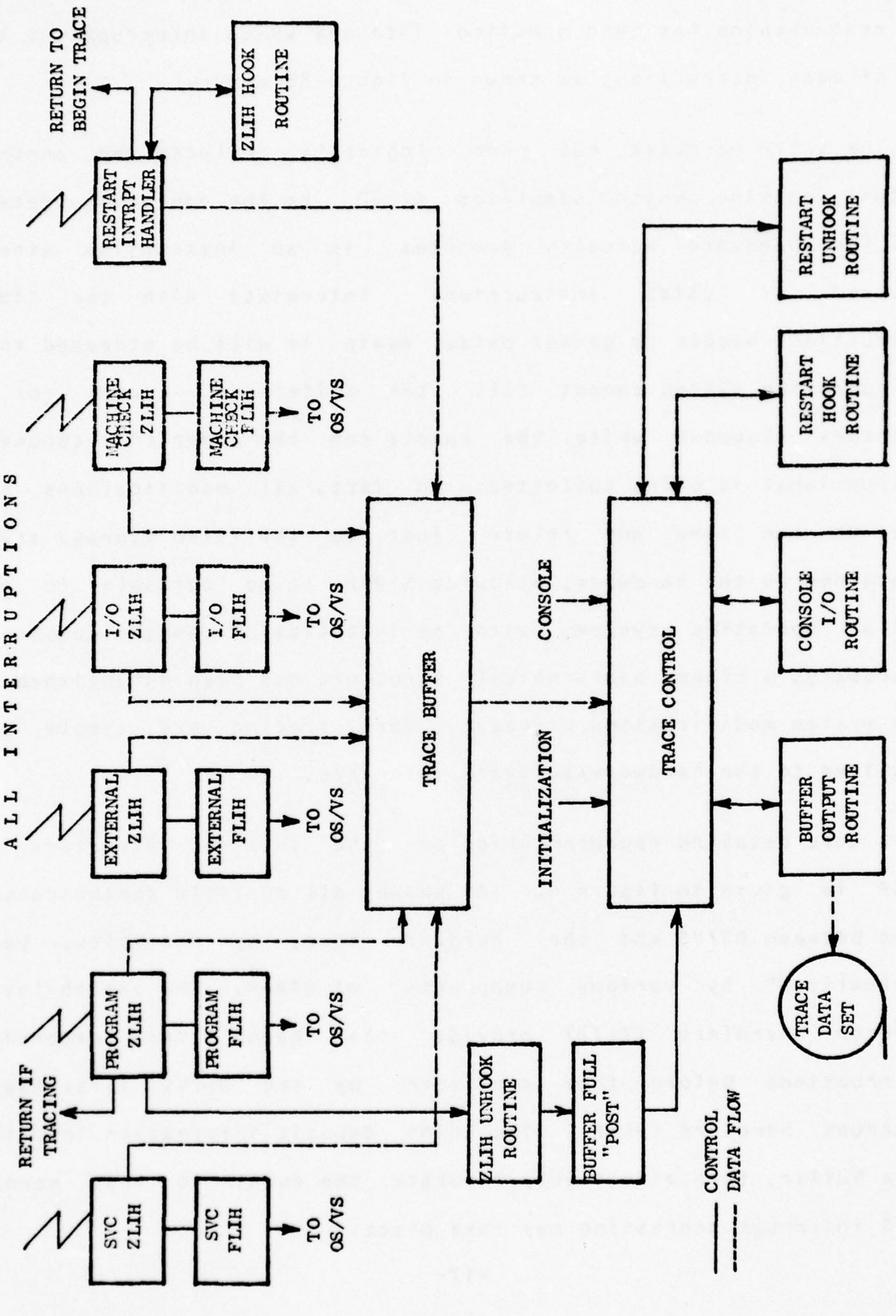

Figure 4. Data and Control Paths.

The ZLIH for program interruptions is markedly different from the others. Since the hardware signals PER events by causing program interruptions, the program interruption ZLIH has the responsibility of recording the relevant data from instruction execution. When it is entered it gathers instruction and address information, then returns control to the next instruction. Because PER is enabled for instruction-fetch trapping over all of the virtual address space, a program interruption will be set up during the execution of that instruction. When it completes, the program interruption ZLIH will again be entered, and the process repeats.

The Restart interrupt handler is responsible for the activation of all ZLIHs upon receipt of an Restart interrupt. It insures that indeed a trace buffer is available, records header information, and constructs the "*traced* virtual machine" discussed earlier by appropriately modifying the new and old PSWs in fixed storage. From that point on, instructions and interruptions are continuously traced until the buffer is filled. The ZLIH UNHOOK routine then restores the environment in which OS/VS normally executes by deactivating the ZLIHs, completes header information in the buffer, and awakens the trace control routine to indicate that a buffer is ready for output. Trace control then schedules the buffer for output to tape or disk and prepares a new buffer for the next trace sample which will begin when the Trace Ace ticks.

As shown in the diagram, facilities are also provided for communication with the operator. He can suspend and restart tracing, display summary information, and perform various control functions related to STRAP. Moreover, the trace control routines run in user mode. Hence modifications can be made and tested using the input/output and error recovery support of OS/VS. In other words, the operating system will continue to function during checkout of new code in the trace control sections. Unfortunately, the same cannot always be said for modifications made to the low-level data gathering routines!

Since STRAP is entirely event driven, a state transition diagram can be used to summarize its operation. The four states that STRAP can be in are:

1. OFF STATE
   a. There are no modifications of OS/VS or its data areas; tracing is not possible.
   b. All new PSWs are "owned" by OS/VS.
   c. The Trace Ace is off.

2. DORMANT-READY STATE
   a. Tracing will begin as soon as the Trace Ace ticks.
   b. STRAP owns the Restart new PSW, OS/VS owns all others.
   c. The Trace Ace is (typically) on.
   d. A trace buffer is available to be filled.

3. DORMANT-NOT-READY STATE

Same conditions as DORMANT-READY execpt that no buffer is available. A Trace Ace tick will cause no effect.

4. ACTIVE (TRACING) STATE

a. Events are being recorded as they occur.

b. All new PSWs are owned by STRAP.

c. PER is enabled for all events and over all of virtual storage.

d. A Trace Ace tick has recently occurred.

e. The S/370 appears to be executing at approximately 45 to 1 slowdown.

The state transition diagram is shown below, along with a diagram illustrating the synchronization of the various STRAP processes.

PRELIMINARY RESULTS

This section presents some examples of the information that can be obtained by tracing. The particular examples given here deal with supervisor usage, instruction usage, and the behavior of supervisor and problem programs with respect to the distance between successful branch executions. The measurements displayed here are from the same monitor run, a 15 minute session during production processing at the Stanford Linear Accelerator Center triplex multiprocessor. The data represents approximately 100,000 instructions and consists of a primarily scientific workload.

Table 1 is a summary of usage of various functions of the OS/VS2 supervisor. It was generated by accumulating dynamic

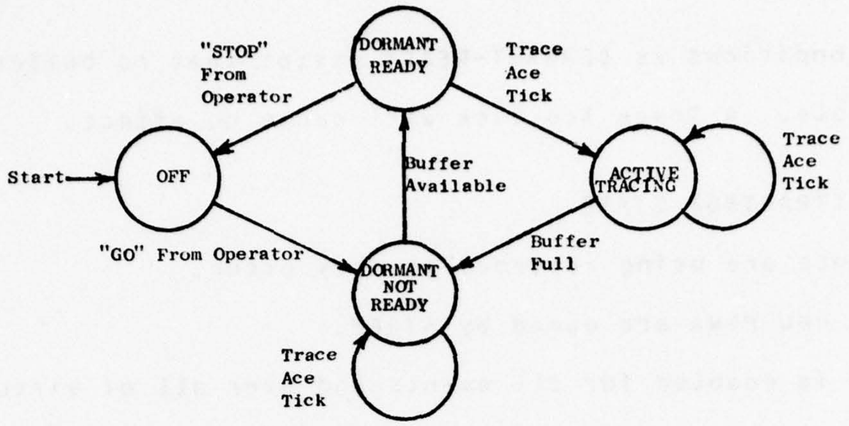

Figure 5. States of the Trace.

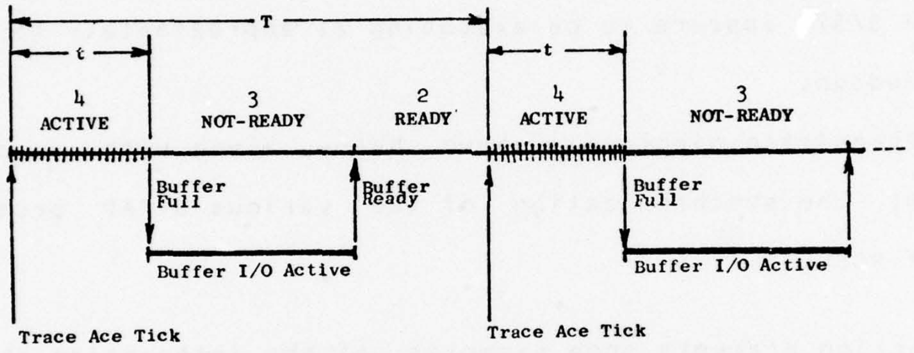

Figure 6. Synchronization of Trace Routines.

instruction execution counts in a histogram representing the OS/VS2 supervisor nucleus. Then a map of the modules within the nucleus was used to coalesce the counts into totals which represent various functions provided by the supervisor, e.g. storage supervision, I/O supervision, etc. In the table they are ranked in decreasing order to demonstrate the relative usage of the respective services. Some interesting characteristics of this particular run are:

1) The I/O supervisor executed almost three times as much code as its nearest competitor, the paging supervisor, and alone it accounted for over one third of all nucleus execution.

2) Over two thirds of all nucleus computing was restricted to four areas: I/O supervisor, paging supervisor, storage supervisor, and the dispatcher.

3) Although the paging rates during the tracing period are known to have been quite low, the paging supervisor exhibited a non-trivial amount of computing. This is most probably due to the fixed overhead required just to maintain the paging system, regardless of the actual amount of paging done.

4) The OS/VS trace routines, which maintain a trace table for diagnostic purposes, accounted for over 8% of nucleus execution.

In all, the results tend to substantiate and quantify intuitive feelings about the behavior of the OS/VS2 supervisor. Further measurement in this area may be helpful in the design of future supervisory programs.

The next two examples show how the trace data can be separated into problem state and supervisor state streams to enable comparisons between supervisor and user programs. Figure 7 shows two frequency histograms which represent the distribution of inter-branch distances. The height of the bar at each bin represents the percentage of occurrence of that particular "run" length between two instruction address counter discontinuities.

TABLE 1. SUPERVISOR USAGE MAP - IN RANK ORDER
(Nucleus Control Sections Only)

Component of OS/VS2 Nucleus	No. Of Instructions Traced	Pct. Of Nucleus Total	Cumu-lative Percent
1. Input/Output Supervisor (IOS)	13,870	34.39 %	34.39 %
2. Paging Supervisor	5,174	12.83	47.22
3. Storage Supervisor	4,437	11.00	58.22
4. Task Dispatcher	3,880	9.62	67.84
5. Contents Management Routines (Pgm. Fetch, etc.)	3,593	8.91	76.75
6. Trace Routines (within OS/VS itself)	3,252	8.06	84.82
7. Wait/Post Routines	2,313	5.74	90.55
8. Interrupt Handlers (FLIHs)	1,495	3.71	94.26
9. Timer Handling Routines	1,164	2.89	97.14
10. Miscellaneous (each < 1.0 percent)	1,152	2.86	100.00
Total Number of Instructions Traced in Nucleus	40,330	100.00	

The curves passing through the histograms are smoothed plots of the same frequency data. These graphs clearly show that the supervisor tends to execute code in shorter bursts, as might be expected. Its purpose is more to control processes and machine resources than to process data and solve problems.

Finally, instruction frequencies are tabulated from the segregated data (Tables 2 and 3). The first two columns of numbers give each instruction's percentage of all instructions executed in that state, and the accumulated percentage, respectively. The PERCENT CHANGE column indicates how the percentage for each instruction differs between the problem and supervisor states. Large differences could indicate dramatic differences in their instruction usage.

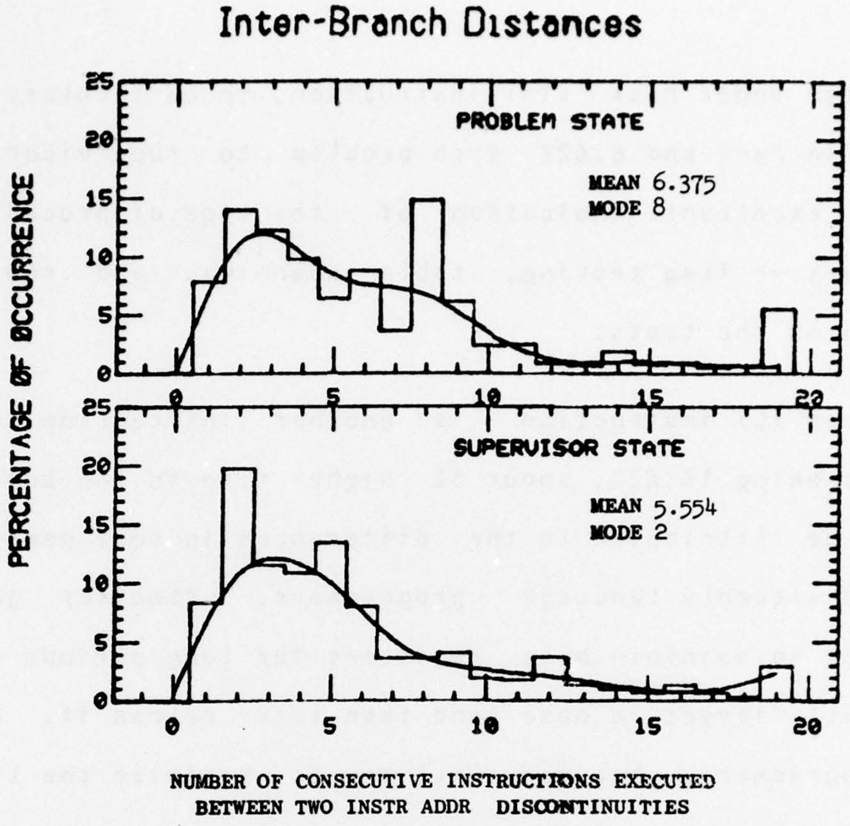

Figure 7.

As might be expected, a differentiation was evident. Some of our observations are discussed here:

1) The supervisor consistently used branch, bit, and character manipulation instructions more that problem programs did. On the other hand, the problem state percentages for the computational instructions (e.g. add, subtract, compare, and floating point) were consistently higher than their supervisor state counterparts.

2) The Test Under Mask (TM) instruction, in particular, climbed 14 positions in rank and 6.62% from problem to supervisor state. This is an excellent indication of the type of processing the supervisor does -- flag testing, table scanning, and routing of control based on the tests.

3) The Load (L) instruction is another interesting example, problem state being 15.62%, about 5% higher than in the supervisor. This could be attributed to the differences in code generated by compilers and assembly language programmers. Compiler generated code tends not to maintain base registers for long periods of time, but rather will "forget" a base and then later reload it. Assembly language programmers, however, will try to maximize the life of a base register, thereby avoiding reloading it later.

CONCLUSIONS

The concept of "virtualizing" the hardware/operating system interface has proven to be useful in tracing global system execution. The ability to collect such detailed information from both user and monitor execution has helped in the further understanding of both, especially where the two modes differ. More extensive use of these techniques will provide a more useful base for the design of future computing systems. The data can now represent all execution on the processor, including the single largest user of computer time, the operating system itself.

Table 2. Supervisor Instruction Freqs. (greater than 1%)

RANK	INSTRUCTION NAME		PERCENT	CUMULATIVE PERCENT	PERCENT CHANGE[1]
1.	BC	Branch on Condn	21.30	21.30	+8.54
2.	L	Load (RX)	10.45	31.75	-5.17
3.	TM	Test Under Mask	7.94	39.69	+6.62
4.	LA	Load Address	5.98	45.67	+3.27
5.	BCR	Branch on Condn	5.55	51.22	+2.82
6.	LR	Load (RR)	5.06	56.28	-1.08
7.	ST	Store	2.86	59.14	-4.74
8.	LTR	Load and Test (RR)	2.57	61.71	+1.16
9.	BALR	Branch and Link (RR)	2.31	64.02	+1.78 *
10.	C	Compare (RX)	2.12	66.14	-0.35
11.	SR	Subtract (RR)	2.08	68.22	-2.10
12.	BAL	Branch and Link (RX)	1.88	70.10	+1.07 *
13.	LM	Load Multiple	1.75	71.85	+0.73
14.	MVC	Move	1.74	73.59	+0.69
15.	STM	Store Multiple	1.69	75.28	+1.06 *
16.	CLI	Compare Log. Immed.	1.47	76.75	+0.77 *
17.	IC	Insert Character	1.38	78.13	+0.52 *
18.	NI	AND Immediate	1.38	79.51	+1.31 *
19.	OI	OR Immediate	1.35	80.86	+1.29 *
20.	ICM	Insert Char. (Mask)	1.34	82.20	+1.34 *
21.	LH	Load Halfword	1.34	83.54	+0.84 *
22.	N	AND (RX)	1.24	84.78	+0.60 *

Notes:

*: Indicates that the instruction occurred less than 1% in problem program state for this run.

1. Difference is expressed as PCT_SUP_STATE - PCT_PROB_STATE for each instruction.

ACKNOWLEDGEMENTS

We would like to express our sincere appreciation to the SLAC Computing Services operations and systems staffs for their incessant cooperation through the development of the system. Also to Dr. R. McClure, B. Rau, and Dr. G. Rossman of Palyn Associates for the Trace Ace, along with their many ideas and

Table 3.

Problem State Instruction Freqs. (greater than 1%)

RANK	INSTRUCTION NAME		PERCENT	CUMULATIVE PERCENT	PERCENT CHANGE[1]
1.	L	Load (RX)	15.62	15.62	+5.17
2.	BC	Branch on Condn	12.75	28.38	-8.54
3.	ST	Store	7.60	35.97	+4.74
4.	LR	Load (RR)	6.14	42.12	+1.08
5.	AR	Add (RR)	5.54	47.65	+4.57 *
6.	SR	Subtract (RR)	4.19	51.84	+2.10
7.	SLL	Shift Left Logical	4.09	55.93	+3.38 *
8.	BXLE	Branch Index LE	3.69	59.62	+3.60 *
9.	LE	Load (Floating)	2.92	62.54	+2.92 *
10.	BCR	Branch on Condn	2.73	65.27	-2.82
11.	LA	Load Address	2.71	67.98	-3.27
12.	C	Compare (RX)	2.47	70.46	+0.35
13.	STE	Store (Floating)	2.47	72.92	+2.47 *
14.	CR	Compare (RR)	2.44	75.37	+1.54 *
15.	A	Add (RX)	2.00	77.36	+1.69 *
16.	LTR	Load and Test (RR)	1.41	78.78	-1.16
17.	TM	Test Under Mask	1.32	80.10	-6.62
18.	MVC	Move	1.04	81.14	-0.69
19.	LM	Load Multiple	1.02	82.16	-0.73

Notes:
***: Indicates that the instruction occurred less than 1% in supervisor state for this run.**

1. Difference is expressed as PCT_PROB_STATE - PCT_SUP_STATE for each instruction.

suggestions. And, finally, to M. Powell for his excellent assistance and the many hours he donated to the implementation of STRAP.

REFERENCES

1. Cheng, P. S., "Trace-driven system modelling," IBM Systems J. 8, 4, 1969, 280-289.

2. Mattson, R. L., Gecsei, J., Slutz, D. R., Traiger, I. L., "Evaluation techniques for storage hierarchies," IBM Systems J. 9, 2, 1970, 78-117.

3. Sherman, S., Baskett, F., Browne, J. C., "Trace-Driven Modeling and Analysis of CPU Scheduling in a Multiprogramming System," Comm. ACM 15, 12, December 1972, 1063-1069.

4. Winder, R. O., "A Data Base for Computer Performance Evaluation," Computer, Vol. 6, No. 3, March 1973, 25-29.

5. Lunde, A., "Evaluation of Instruction Set Processor Architecture by Program Tracing," Ph. D. thesis, Carnegie-Mellon University, 1974.

6. "Users Guide to Triplex System," SLAC Computing Services, User Note 69, July 1974.

7. "OS/VS System Management Facilities (SMF)," IBM Systems Reference Library, GC35-0004, 1973.

8. "IBM OS/VS Service Aids," IBM Systems Reference Library, GC28-0633, 1973.

9. Holtwick, G., "Designing a Commercial Performance Measurement System," Proc. ACM SIGOPS Workshop on System Performance Evaluation, April 1971, 29-58.

10. Cantrell, H. N. and Ellison, A. L., "Multiprogramming system performance and analysis," Proc. AFIPS 1968 SJCC, Vol. 32, 213-221.

11. "IBM System/370 Principles of Operation," IBM Systems Reference Library, GA22-7000, 1973.

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
Technical Note No. 97

2. GOVT ACCESSION NO.
14 DSL-TN-97

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
6 The Design and Implementation of an Operating System Tracer.

5. TYPE OF REPORT & PERIOD COVERED
9 Technical note

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
10 David J. Rossetti [redacted] Thomas H. Bredt

8. CONTRACT OR GRANT NUMBER(s)
JSEP N00014-67-A-0112-0044 15

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Digital Systems Laboratory
Stanford University
Stanford, CA 94305

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
7151

11. CONTROLLING OFFICE NAME AND ADDRESS
Sponsored Projects Office
Stanford University
Stanford, California 94305

12. REPORT DATE
11 Oct[redacted]77

13. NO. OF PAGES
30 12 33p.

14. MONITORING AGENCY NAME & ADDRESS (if diff. from Controlling Office)

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

computer architecture	operating systems
instruction set design,	performance evaluation
IBM 370	tracing
measurement	virtual machine

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

STRAP, a general system for collecting a trace of machine-instruction execution on a high performance processor, has been implemented. Its generality lies in its ability to trace all instructions executed by the processor, whether they are issued by the supervisor or by a user program, and also in its ability to do this without significant impact on system performance. STRAP creates a 'virtual machine' environment in which the operating system appears to be executing normally, but is actually having its instructions traced by a program lying between it and the real processor. Burst sampling is used to avoid excessive tracing overhead, and the

over

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

408071

TC

20. ABSTRACT (continued)

real processor interprets the traced instructions, keeping the instantaneous burst overhead down to about 45:1.

The resulting traces, which can be useful for architectural studies and performance evaluation of existing systems, can now contain data for all parts of the operating system. We present example results showing how the data have been used to study: 1) The usage of various operating system components, 2) The difference in branching patterns between supervisor and user code, and 3) The difference in instruction frequency distribution between the two modes of operation.

JSEP REPORTS DISTRIBUTION LIST

Department of Defense

Director
National Security Agency
Attn: Dr. T. J. Beahn
Fort George G. Meade
Maryland 20755

Defense Documentation Center (12)
Attn: DDC-TCA (Mrs. V. Caponio)
Cameron Station
Alexandria, Virginia 22314

Assistant Director
Electronics and Computer Sciences
Office of Director of Defense
Research and Engineering
The Pentagon
Washington, D.C. 20315

Defense Advanced Research
Projects Agency
Attn: Dr. R. Reynolds
1400 Wilson Boulevard
Arlington, Virginia 22209

Department of the Army

Commandant
US Army Air Defense School
Attn: ATSAD-T-CSM
Fort Bliss, Texas 79916

Commander
US Army Armament R&D Command
Attn: DRDAR-TSS
Dover, New Jersey 07801

Commander
US Army Armament R&D Command (BRL)
Attn: DRDAR-TSB-S
Aberdeen Proving Ground
Aberdeen, Maryland 21005

Commandant
US Army Command and General Staff
College
Attn: Acquisitions, Library Division
Fort Leavenworth, Kansas 66027

Commander
US Army Communication Command
Attn: CC-OPS-PD
Fort Huachuca, Arizona 85613

Commander
US Army Materials and Mechanics
Research Center
Attn: Chief, Materials Science Div.
Watertown, Massachusetts 02172

Commander
US Army Materiel Development and
Readiness Command
Attn: Technical Library, Rm. 7S 35
5001 Eisenhower Avenue
Alexandria, Virginia 22333

Commander
US Army Missile R&D Command
Attn: Chief, Document Section
Redstone Arsenal, Alabama 35809

Commander
US Army Satellite Communications Agency
Fort Monmouth, New Jersey 07703

Director
US Army Signals Warfare Laboratory
Attn: DELSW-OS
Arlington Hall Station
Arlington, Virginia 22212

Project Manager
ARTADS
EAI Building
West Long Branch, New Jersey 07764

NOTE: One (1) copy to each addressee unless otherwise indicated.

Commander/Director
Atmospheric Sciences Lab. (ECOM)
Attn: DRSEL-BL-DD
White Sands Missile Range
New Mexico 88002

Commander
US Army Electronics Command
Attn: DRSEL-NL-O (Dr. H. S. Bennett)
Fort Monmouth, New Jersey 07703

Director
TRI-TAC
Attn: TT-AD (Mrs. Briller)
Fort Monmouth, New Jersey 07703

Commander
US Army Electronics Command
Attn: DRSEL-CT-L (Dr. R. Buser)
Fort Monmouth, New Jersey 07703

Director
Electronic Warfare Lab. (ECOM)
Attn: DRSEL-WL-MY
White Sands Missile Range
New Mexico 88002

Executive Secretary, TAC/JSEP
US Army Research Office
P. O. Box 12211
Research Triangle Park
North Carolina 27709

Project Manager
Ballistic Missile Defense Program
Office
Attn: DACS-DMP (Mr. A. Gold)
1300 Wilson Boulevard
Arlington, Virginia 22209

Commander
Harry Diamond Laboratories
Attn: Mr. John E. Rosenberg
2800 Powder Mill Road
Adelphi, Maryland 20783

HQDA (DAMA-ARZ-A)
Washington, D.C. 20310

Commander
US Army Electronics Command
Attn: DRSEL-TL-E (Dr. J. A. Kohn)
Fort Monmouth, New Jersey 07703

Commander
US Army Electronics Command
Attn: DRSEL-TL-EN
(Dr. S. Kroenenberg)
Fort Monmouth, New Jersey 07703

Commander
US Army Electronics Command
Attn: DRSEL-NL-T (Mr. R. Kulinyi)
Fort Monmouth, New Jersey 07703

Commander
US Army Electronics Command
Attn: DRSEL-NL-B (Dr. E. Lieblein)
Fort Monmouth, New Jersey 07703

Commander
US Army Electronics Command
Attn: DRSEL-TL-MM (Mr. N. Lipetz)
Fort Monmouth, New Jersey 07703

Commander
US Army Electronics Command
Attn: DRSEL-RD-O (Dr. W. S. McAfee)
Fort Monmouth, New Jersey 07703

Director
Night Vision Laboratory
Attn: DRSEL-NV-D
Fort Belvoir, Virginia 22060

Col. Robert Noce
Senior Standardization Representative
US Army Standardization Group, Canada
Canadian Force Headquarters
Ottawa, Ontario, CANADA KIA)K2

Commander
US Army Electronics Command
Attn: DRSEL-NL-B (Dr. D. C. Pearce)
Fort Monmouth, New Jersey 07703

Commander
US Army Electronics Command
Attn: DRSEL-NL-RH-1
(Dr. F. Schwering)
Fort Monmouth, New Jersey 07703

Commander
US Army Electronics Command
Attn: DRSEL-TL-I
(Dr. C. G. Thornton)
Fort Monmouth, New Jersey 07703

US Army Research Office (3)
Attn: Library
P. O. Box 12211
Research Triangle Park
North Carolina 27709

Director
Division of Neuropsychiatry
Walter Reed Army Institute
of Research
Washington, D.C. 20012

Commander
White Sands Missile Range
Attn: STEWS-ID-R
White Sands Missle Range
New Mexico 88002

<u>Department of the Air Force</u>

Mr. Robert Barrett
RADC/ETS
Hanscom AFB, Massachusetts 01731

Dr. Carl E. Baum
AFWL (ES)
Kirtland AFB, New Mexico 87117

Dr. E. Champagne
AFAL/DH
Wright-Patterson AFB, Ohio 45433

Dr. R. P. Dolan
RADC/ETSD
Hanscom AFB, Massachusetts 01731

Mr. W. Edwards
AFAL/TE
Wright-Patterson AFB, Ohio 45433

Professor R. E. Fontana
Head, Department of Electrical
Engineering
AFIT/ENE
Wright-Patterson AFB, Ohio 45433

Dr. Alan Garscadden
AFAPL/POD
Wright-Patterson AFB, Ohio 45433

USAF European Office of Aerospace
Research
Attn: Major J. Gorrell
Box 14, FPO, New York 09510

LTC Richard J. Gowen
Department of Electrical Engineering
USAF Academy, Colorado 80840

Mr. Murray Kesselman (ISCA)
Rome Air Development Center
Griffiss AFB, New York 13441

Dr. G. Knausenberger
Air Force Member, TAC
Air Force Office of Scientific
Research, (AFSC) AFSOR/NE
Bolling Air Force Base, DC 20332

Dr. L. Kravitz
Air Force Member, TAC
Air Force Office of Scientific
Research, (AFSC) AFSOR/NE
Bolling Air Force Base, DC 20332

Mr. R. D. Larson
AFAL/DHR
Wright-Patterson AFB, Ohio 45433

Dr. Richard B. Mack
RADC/ETER
Hanscom AFB, Massachusetts 01731

Mr. John Mottsmith (MCIT)
HQ ESD (AFSC)
Hanscom AFB, Massachusetts 01731

Dr. Richard Picard
RADC/ETSL
Hanscom AFB, Massachusetts 01731

Dr. J. Ryles
Chief Scientist
AFAL/CA
Wright-Patterson AFB, Ohio 45433

Dr. Allan Schell
RADC/ETE
Hanscom AFB, Massachusetts 01731

Mr. H. E. Webb, Jr. (ISCP)
Rome Air Development Center
Griffisss AFB, New York 13441

LTC G. Wepfer
Air Force Office of Scientific
Research, (AFSC) AFOSR/NP
Bolling Air Force Base, DC 20332

LTC G. McKemie
Air Force Office of Scientific
Research, (AFSC) AFOSR/NM
Bolling Air Force Base, DC 20332

Department of the Navy

Dr. R. S. Allgaier
Naval Surface Weapons Center
Code WR-303
White Oak
Silver Spring, Maryland 20910

Naval Weapons Center
Attn: Code 5515, H. F. Blazek
China Lake, California 93555

Dr. H. L. Blood
Technical Director
Naval Undersea Center
San Diego, California 95152

Naval Research Laboratory
Attn: Code 5200, A. Brodzinsky
4555 Overlook Avenue, SW
Washington, D.C. 20375

Naval Research Laboratory
Attn: Code 7701, J. D. Brown
4555 Overlook Avenue, SW
Washington, D.C. 20375

Naval Research Laboratory
Attn: Code 5210, J. E. Davey
4555 Overlook Avenue, SW
Washington, D.C. 20375

Naval Research Laboratory
Attn: Code 5460/5410, J. R. Davis
4555 Overlook Avenue, SW
Washington, D.C. 20375

Naval Ocean Systems Center
Attn: Code 75, W. J. Dejka
271 Catalina Boulevard
San Diego, California 92152

Naval Weapons Center
Attn: Code 601, F. C. Essig
China Lake, California 93555

Naval Research Laboratory
Attn: Code 5510, W. L. Faust
4555 Overlook Avenue, SW
Washington, D.C. 20375

Naval Research Laboratory
Attn: Code 2626, Mrs. D. Folen
4555 Overlook Avenue, SW
Washington, D.C. 20375

Dr. Robert R. Fossum
Dean of Research
Naval Postgraduate School
Monterey, California 93940

Dr. G. G. Gould
Technical Director
Naval Coastal System Laboratory
Panama City, Florida 32401

Naval Ocean Systems Center
Attn: Code 7203, V. E. Hildebrand
271 Catalina Boulevard
San Diego, California 92152

Naval Ocean Systems Center
Attn: Code 753, P. H. Johnson
271 Catalina Boulevard
San Diego, California 92152

Donald E. Kirk
Professor and Chairman
Electronic Engineer, SP-304
Naval Postgraduate School
Monterey, California 93940

Naval Air Development Center
Attn: Code 01, Dr. R. K. Lobb
Johnsville
Warminster, Pennsylvania 18974

Naval Research Laboratory
Attn: Code 5270, B. D. McCombe
4555 Overlook Avenue, SW
Washington, D.C. 20375

Capt. R. B. Meeks
Naval Sea Systems Command, NC #3
2531 Jefferson Davis Highway
Arlington, Virginia 20362

Dr. H. J. Mueller
Naval Air Systems Command
Code 310, JP #1
1411 Jefferson Davis Highway
Arlington, Virginia 20360

Dr. J. H. Mills, Jr.
Naval Surface Weapons Center
Electronics Systems Department
Code DF
Dahlgren, Virginia 22448

Naval Ocean Systems Center
Attn: Code 702, H. T. Mortimer
271 Catalina Boulevard
San Diego, California 92152

Naval Air Development Center
Attn: Technical Library
Johnsville
Warminster, Pennsylvania 18974

Naval Ocean Systems Center
Attn: Technical Library
271 Catalina Boulevard
San Diego, California 92152

Naval Research Laboratory
Underwater Sound Reference Division
Technical Library
P. O. Box 8337
Orlando, Florida 32806

Naval Surface Weapons Center
Attn: Technical Library
Code DX-21
Dahlgren, Virginia 22448

Naval Surface Weapons Center
Attn: Technical Library
Building 1-330, Code WX-40
White Oak Laboratory
Silver Spring, Maryland 20910

Naval Training Equipment Center
Attn: Technical Library
Orlando, Florida 32813

Naval Undersea Center
Attn: Technical Library
San Diego, California 92152

Naval Underwater Systems Center
Attn: Technical Library
Newport, Rhode Island 02840

Office of Naval Research
Electronic and Solid State
Sciences Program (Code 427)
800 North Quincy Street
Arlington, Virginia 22217

Office of Naval Research
Mathematics Program (Code 432)
800 North Quincy Street
Arlington, Virginia 22217

Office of Naval Research
Naval Systems Division
Code 220/221
800 North Quincy Street
Arlington, Virginia 22217

Director
Office of Naval Research
New York Area Office
715 Broadway, 5th Floor
New York, New York 10003

Office of Naval Research
San Francisco Area Office
One Hallidie Plaza, Suite 601
San Francisco, California 94102

Director
Office of Naval Research Branch Office
495 Summer Street
Boston, Massachusetts 02210

Director
Office of Naval Research Branch Office
536 South Clark Street
Chicago, Illinois 60605

Director
Office of Naval Research Branch Office
1030 East Green Street
Pasadena, California 91101

Mr. H. R. Riedl
Naval Surface Weapons Center
Code WR-34
White Oak Laboratory
Silver Spring, Maryland 20910

Naval Air Development Center
Attn: Code 202, T. J. Shopple
Johnsville
Warminster, Pennsylvania 18974

Naval Research Laboratory
Attn: Code 5403, J. E. Shore
4555 Overlook Avenue, SW
Washington, D.C. 20375

A. L. Slafkovsky
Scientific Advisor
Headquarters Marine Corps
MC-RD-1, Arlington Annex
Washington, D.C. 20380

Harris B. Stone
Office of Research, Development,
Test and Evaluation
NOP-987
The Pentagon, Room 5D760
Washington, D.C. 20350

Mr. L. Sumney
Naval Electronics Systems Command
Code 3042, NC #1
2511 Jefferson Davis Highway
Arlington, Virginia 20360

David W. Taylor
Naval Ship Research and
Development Center
Code 522.1
Bethesda, Maryland 20084

Naval Research Laboratory
Attn: Code 4105, Dr. S. Teitler
4555 Overlook Avenue, SW
Washington, D.C. 20375

Lt. Cdr. John Turner
NAVMAT 0343
CP #5, Room 1044
2211 Jefferson Davis Highway
Arlington, Virginia 20360

Naval Ocean Systems Center
Attn: Code 746, H. H. Wieder
271 Catalina Boulevard
San Diego, California 92152

Dr. W. A. Von Winkle
Associate Technical Director for
Technology
Naval Underwater Systems Center
New London, Connecticut 06320

Dr. Gernot M. R. Winkler
Director, Time Service
US Naval Observatory
Massachusetts Ave. at 34th St., NW
Washington, D.C. 20390

Other Government Agencies

Dr. Howard W. Etzel
Deputy Director
Division of Materials Research
National Science Foundation
1800 G Street
Washington, D.C. 20550

Mr. J. C. French
National Bureau of Standards
Electronics Technology Division
Washington, D.C. 20234

Dr. Jay Harris
Program Director
Devices and Waves Program
National Science Foundation
1800 G Street
Washington, D.C. 20550

Los Alamos Scientific Laboratory
Attn: Reports Library
P. O. Box 1663
Los Alamos, New Mexico 87544

Dr. Dean Mitchell
Program Director
Solid-State Physics
Division of Materials Research
National Science Foundation
1800 G Street
Washington, D.C. 20550

Mr. F. C. Schwenk, RD-T
National Aeronautics and Space
Administration
Washington, D.C. 20546

M. Zane Thornton
Deputy Director, Institute for
Computer Sciences and Technology
National Bureau of Standards
Washington, D.C. 20234

Nongovernment Agencies

Director
Columbia Radiation Laboratory
Columbia University
538 West 120th Street
New York, New York 10027

Director
Coordinated Science Laboratory
University of Illinois
Urbana, Illinois 61801

Director of Laboratories
Division of Engineering and
Applied Physics
Harvard University
Pierce Hall
Cambridge, Massachusetts 02138

Director
Electronics Research Center
The University of Texas
Engineering-Science Bldg. 112
Austin, Texas 78712

Director
Electronics Research Laboratory
University of California
Berkeley, California 94720

Director
Electronics Sciences Laboratory
University of Southern California
Los Angeles, California 90007

Director
Microwave Research Institute
Polytechnic Institute of New York
333 Jay Street
Brooklyn, New York 11201

Director
Research Laboratory of Electronics
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Director
Stanford Electronics Laboratories
Stanford University
Stanford, California 94305

Stanford Ginzton Laboratory
Stanford University
Stanford, California 94305

Officer in Charge
Carderock Laboratory
Code 18, G. H. Gleissner
David Taylor Naval Ship Research
and Development Center
Bethesda, Maryland 20084

Dr. Roy F. Potter
3868 Talbot Street
San Diego, California 92106